



**CENTRE
DE RECHERCHE
SUR LES TRANSPORTS
(CRT)**

**CENTRE
FOR RESEARCH
ON TRANSPORTATION**

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE



C.P. 6128, SUCCURSALE A
MONTRÉAL (QUÉBEC)
CANADA
H3C 3J7

TÉLÉPHONE : (514) 343-7575
TÉLEX : 05-24146 BIBPOLYTEC
TÉLÉCOPIEUR (FAX) : (514) 343-7121

Published in :

"Proceedings of Technology of Object-Oriented Language and Systems",
Bézivin, J. (ed), Paris, 1989, 325-335.

CRT-670

**AN APPLICATION OF OBJECT-ORIENTED PROGRAMMING
FOR MODELING AND SOLVING VEHICLE ROUTING PROBLEMS**

by

Jean-Yves Potvin¹

Guy Lapalme¹

Jean-Marc Rousseau¹

¹ Centre de recherche sur les transports, Université de Montréal, C.P. 6128,
Succursale "A", Montréal, Québec, Canada H3C 3J7.

This research was supported by a grant from F.C.A.R. - Actions spontanées,
and by the Natural Sciences and Engineering Research Council of Canada.

90-01604/

Dépôt légal - 4e trimestre 1989

Abstract. This paper describes an application of the object-oriented paradigm for a vehicle routing computer system aimed at facilitating the design of routing algorithms. The object-oriented paradigm looked very attractive in such a context, because vehicle routing problems can be easily decomposed into well-defined entities, like the network with its nodes and links, the routes and the vehicles. The problem-solving procedure (or general heuristic) is modeled accordingly as a set of generic operators that are applied to the symbolic objects describing the problem to be solved.

Key Words. Object-Oriented Programming, General Heuristic, Vehicle Routing

Résumé. Cet article décrit une application de la programmation orientée objet pour le développement d'un système informatique facilitant la conception d'algorithmes de tournées. L'approche orientée objet apparaît particulièrement intéressante ici, puisque tout problème de tournées se décompose en entités distinctes, tels le réseau avec ses noeuds et ses liens, les routes générées et les véhicules desservant ces routes. L'approche de résolution (ou heuristique générale) est constituée quant à elle d'un ensemble d'opérateurs génériques qui manipulent et modifient les objets symboliques décrivant le problème à résoudre.

Mots Clés. Programmation Orientée Objet, Heuristique Générale, Génération de Tournées.

An Application of Object-Oriented Programming for Modeling and Solving Vehicle Routing Problems

Abstract. This paper describes an application of the object-oriented paradigm for a vehicle routing computer system aimed at facilitating the design of routing algorithms. The object-oriented paradigm looked very attractive in such a context, because vehicle routing problems can be easily decomposed into well-defined entities, like the network with its nodes and links, the routes and the vehicles. The problem-solving procedure (or general heuristic) is modeled accordingly as a set of generic operators that are applied to the symbolic objects describing the current routing problem.

Section 1. Introduction

The aim of this paper is to describe an application of the object-oriented paradigm for an interactive and graphic system, called ALTO, aimed at facilitating the design of heuristic algorithms in the vehicle routing domain. ALTO is written in LOOPS and is implemented on the 1108 Xerox Lisp Machine which provides a rich and powerful environment known as INTERLISP-D [6]. Since the problem-solving procedure is based on a "general heuristic" instantiated by an expert user with his/her own formulas so as to guide the system towards "good" solutions, the interactive and graphic facilities of INTERLISP-D are of prime importance. Within the current environment, the expert user can easily interact with the system during the problem-solving process, because it is possible for him/her to see how the routes are constructed, based on his/her instantiation formulas, and to revise these formulas at any time if they appear unsatisfactory.

ALTO is really an integrated system providing graphic facilities for the display of transportation networks, interactive facilities allowing specification of routing problems and algorithmic facilities for solving the problems so defined. It has already been successfully applied to "real-life" vehicle routing problems, for which no exact algorithmic approaches exist (cf. experiments with real mail pick-up problems, as described in [3]).

Since the full description of the general heuristic is the subject of another paper and is really related to operations research [2], we will mainly turn our attention towards the internal structure of the system which is organized as a hierarchical set of routing objects. Section 2 will first characterize the class of routing problems for which ALTO has been designed. Then, section 3 will describe the main objects within the system, their hierarchical relationships and how they are used in the problem specification. Section 4 then gives a short description of the general

heuristic and explains how it exploits the object representation. Finally, section 5 draws general conclusions about the benefits of the object-oriented approach in our application domain.

Section 2. The target class of problems

The current system is designed for a class of transportation problems known as "vehicle routing problems". Since many different kinds of such problems are described in the literature [1], we restricted somewhat the field of application of ALTO in the current implementation. To this end, we took apart well-known characteristics that are found in most realistic applications. These characteristics are:

- a) An homogeneous or heterogeneous fleet of vehicles is housed at a central origin (depot).
- b) A set of stops (clients) must be serviced by the fleet of vehicles.
- c) Some quantity of a product (demand) is collected at each stop. This demand must be collected by a single vehicle.
- d) We consider one or more of the following constraints:
 - i) capacity constraints for each vehicle (i.e. the demand collected by a vehicle should not exceed its capacity)
 - ii) maximal distance or travel time for each vehicle
 - iii) time windows at each stop (i.e. each stop must be serviced within some time interval)
- e) The objective is to optimize the routes under the above constraints with respect to various qualitative and quantitative criteria, like the length of the routes, the quality of the service time at the stops, etc...

ALTO was designed with this restricted class of vehicle routing problems in mind and the next section will now present the various objects modeling such problems.

Section 3. The object-oriented paradigm

We will first briefly describe the main features of LOOPS, the implementation language of ALTO. Then, we will present the internal organization of the routing objects within the system.

3.1 LOOPS : An Object-Oriented Programming Language

We must first observe that LOOPS considers two different kinds of objects: classes and instances. A class is a description of one or more similar objects while an instance refers to a particular occurrence of a given class. Objects combine the properties of procedures and data: they store data in variables and respond to messages by carrying out procedures (methods). Variables are divided in two categories: class variables (CV) are used to hold informations that are shared by all instances of a given class while instance variables (IV) contain informations relating to a particular instance. Each instance privately holds the specific values of its instance variables. A class, on the other hand, contains the class variables, methods and instance variable names (with optional "default values").

The classes so defined are then combined so as to create multiple inheritance lattices allowing incremental specialization of their basic components. "Mixins" (non instantiable classes) are also provided in order to augment the description and behavior of instantiable classes. A mixin is thus a special kind of class that encapsulates variables and methods and provide new properties to one or more subclasses.

In the following, we will refer freely to the basic terminology introduced above.

3.2 The classes of objects within ALTO

The organization of the program as a set of related objects stems naturally from our application domain. In fact, any vehicle routing problem is composed of well-defined entities such as a network with nodes, links and streets, an origin, a set of stops, a fleet of vehicles and a set of routes. Furthermore, in most routing applications, some nodes or links carry a special meaning. For example, with respect to our target class of problems, a set of stops and an origin must be located on the network, and this can be done by specializing the basic nodes (or simple nodes) of the underlying transportation network. This led us to the design of the multiple inheritance lattice shown in figure 1.

As we can see, six subclasses of the primitive class ROUTING-OBJECT are defined: NETWORK, NODE, LINK, STREET, VEHICLE and ROUTE. The class NODE is then divided into the specialized classes SIMPLE-NODE and SPECIAL-NODE. Simple nodes are located where the links of the network intersect (in urban networks, such intersections are the corners of the streets), while a special node is either a STOP or an ORIGIN. Some mixins - the unboxed classes in figure 2- are also included in the lattice in order to add some specific properties to those basic classes. For example, NETWORK-DISPLAY contains variables and methods for the display of the network, NETWORK-INTERFACE controls some specific input/output functions while NETWORK-DIJKSTRA, NETWORK-HEURISTIC and NETWORK-TS (for Traveling Salesman, see [1]) deal with the implementation of the various routing algorithms.

Many other interrelations among the routing objects exist beside the hierarchical links shown in figure 1. For example, a network is a composite object made of nodes and links. Hence, any instance of the class NETWORK contains two variables, NODE-LIST and LINK-LIST, holding pointers to the appropriate node and link instances. In the same way, a STREET is a composite object made of links (each link being a portion of street that extends between two intersections).

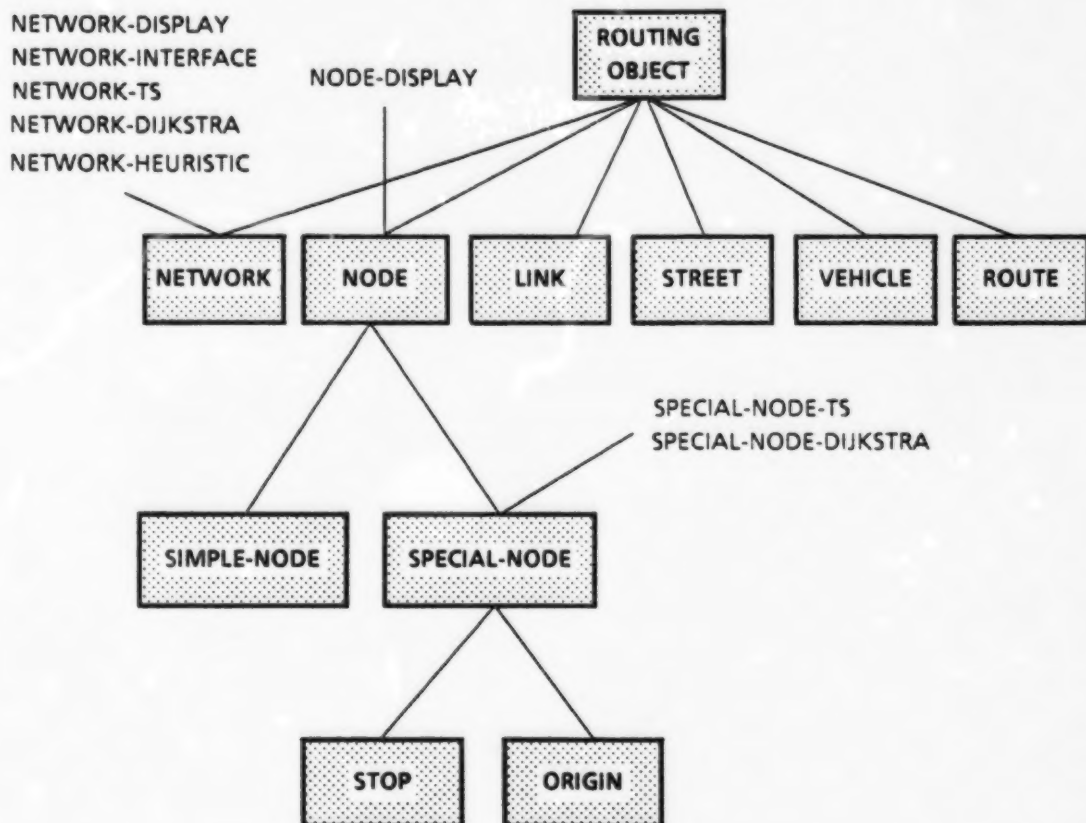


Fig. 1 Classes of objects of ALTO

3.3 Modeling Vehicle Routing Problems with Symbolic Objects

A vehicle routing problem is modeled within ALTO via instances of the above classes of object. More precisely, specifying a problem with N stops and M vehicles for the general heuristic requires the following objects and attributes:

a) one instance of the class ORIGIN with the following attributes:

- coordinates : location of the origin on the network
- fleet-of-vehicles : list of vehicles that are housed within the origin
- distance vector : distance to the other stops

We should note here that the distance vectors are generated using the object-oriented description of the underlying transportation network. This description is done via instances of the following classes of objects: SIMPLE-NODE, LINK and STREET. Hence, these objects and the relationships between them define the structure of the transportation network which is used to evaluate the shortest paths between the stops. These object are not used as such in the specification of the vehicle routing problem.

b) N instances of the class STOP with the following attributes:

- coordinates : location of the stop on the network
- demand : quantity of product to be collected
- loading time : time needed to collect the demand and load the vehicle
- time window : time interval for service at the stop
- distance vector : distance to the origin and to the other stops

c) M instances of the class VEHICLE with the following attributes:

- speed
- capacity
- maximal distance
- maximal travel time

d) M instances of the class ROUTE. A solution to a vehicle routing problem is in fact a set of routes. Hence, these routes are initially empty and the value of their attributes are initialized and modified as the resolution process goes on. The routes are described via the following attributes:

- list-of-stops : stops that are included in the route
- total-length : total length of the route
- total-time : total time for servicing the route
- total-demand : total demand to be collected on the route
- vehicle : vehicle servicing the route

Instances of the classes ORIGIN, STOP and VEHICLE are created and initialized by the user, using facilities for interactive creation, inspection and modification of routing objects. The routes, however, are not directly accessible to the user and are created and modified by the system as the problem-solving process goes on.

The next section will not explain how this process of instantiating classes of objects is used to create resolution strategies that are then applied to the current vehicle routing problem.

Section 4. A General Heuristic for Vehicle Routing Problems

The resolution procedure of ALTO is called the "general heuristic" and represents a general framework for the specification of new resolution strategies. More precisely, the general heuristic is composed of three generic operators (or problem-solving classes) that are instantiated and applied to the current set of routing objects. These generic operators are in fact templates modeling general problem-solving procedures in the vehicle routing domain, and these templates are instantiated by the user with his/her own formulas so as to create specific instances (i.e. specific problem-solving procedures). It is possible to create many different instances of any given generic operator and then to combine all these specific procedures into a complete resolution strategy. The application of a single problem-solving instance is thus not sufficient to obtain a final solution to the problem at hand. It rather represents a step towards the final solution.

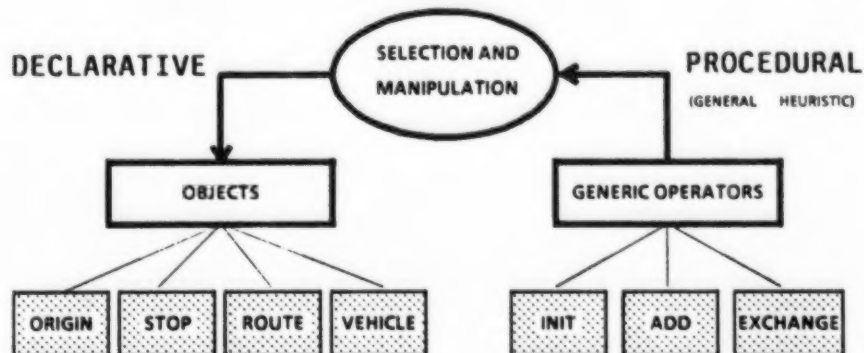


Fig. 2 Objects and generic operators

As we can see from figure 2, the objects defining a given routing problem are selected and manipulated by the following generic operators:

- a) INIT: operator for the initialization of new routes
- b) ADD: operator for the insertion of new stops into existing routes

c) **EXCHANGE:** operator for the resequencing of stops within existing routes

Each operator first selects one or more routing objects and then manipulates and modifies these objects. The basic manipulation process is specified within each generic operator but the selection of appropriate routing objects is controlled by means of user-defined instantiation formulas. For example, the generic behavior of operator ADD is to insert stops within existing routes. A basic loop iterates over all stops not serviced yet by a vehicle and, at each step, one of those stops is selected, a route is selected and the selected stop is inserted within the selected route. This generic structure is instantiated by the user with his/her own formulas (or rules) so as to appropriately select the free stop, the route and the location within the route for the insertion of the stop. User-defined instantiation formulas are thus used to create specific instances of the generic problem-solving classes INIT, ADD and EXCHANGE.

Any specific problem-solving instance can then be applied to the current solution, which is either a partial set of routes that does not yet cover all stops or a complete set of routes that can still be improved via exchange heuristics. If the application of a given problem-solving instance on the current set of routes does not produce good results (i.e. the new set of routes does not look good), it is possible to backtrack to the previous problem state and try something else. ALTO thus represents a very flexible framework for creating and experimenting with new resolution strategies. This is done easily and quickly by creating and modifying instantiation formulas and incorporating them into new problem-solving instances. We showed in previous papers that this framework can model a wide class of well-known heuristic algorithms already documented in the literature [3].

The flexibility of ALTO is very attractive in our application domain because many vehicle routing problems are NP-hard and no exact polynomially-bounded algorithms exist for solving them. Heuristic approaches are thus widely used and ALTO allows experts to easily experiment with various heuristic resolution strategies so as to identify the ones that are best suited to the problems at hand (see [2,3]).

Section 5. Benefits of Object-Oriented Programming

The aim of this section is to emphasize the benefits provided by the object-oriented approach in our application domain. To this end, we must again emphasize the two main features of ALTO. First, the ability to interactively specify new problems by creating, deleting and modifying the routing objects involved in the problem's definition and, secondly, the ability to dynamically solve the problem by applying instances of generic problem-solving classes to the current problem state. We will first present some practical examples of ways to take advantage of the

object-oriented approach. We will then draw some conclusions about the benefits of this approach with respect to the internal structure of the program and with respect to the user interface.

5.1 Practical examples

The following examples illustrate in very practical terms some benefits of the object oriented paradigm for knowledge representation and solution management. We will then draw general observations from these examples.

Example 1. The class NODE and its related subclasses

Since ALTO generates routes for clients located on the nodes of a transportation network, the class NODE and its subclasses deserve a special attention. In figure 3, the node's hierarchy includes some illustrative variables and methods.

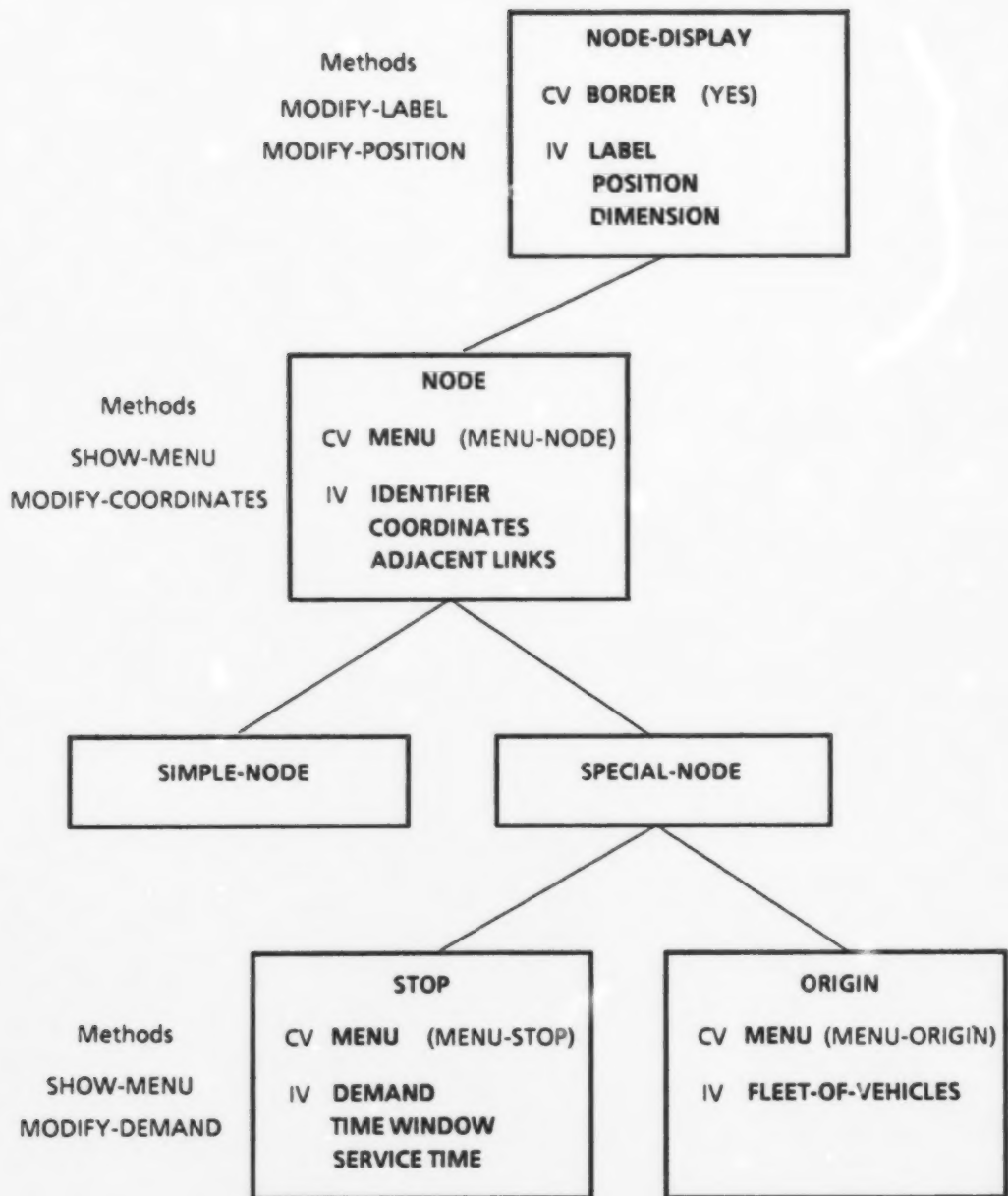


Fig. 3 The class **NODE** and its related classes

This figure shows the incremental specialization of various classes, as we go down from a NODE to a STOP or an ORIGIN. The data stored in these classes are hierarchically organized so as to avoid redundancy. Hence, the class NODE contains general attributes and methods common to all nodes in the transportation network like an identifier, the coordinates, the list of adjacent links and the methods to manipulate such data. The descendants of this class contain more specialized attributes that are related to restricted subsets of nodes in the network. For example, instance variables like DEMAND and TIME-WINDOW in the class STOP are not available to a SIMPLE-NODE or an ORIGIN. In the same way, an ORIGIN is the only type of node concerned with the management of the fleet-of-vehicles.

As mentioned earlier, class variables contain values shared by all instances of a given class. For example, the variable MENU in the class STOP is a CV whose value is MENU-STOP, a specialized menu allowing interactive modifications to the content of any user-selected stop. Selections made within this menu thus trigger methods contained in that class, like MODIFY-DEMAND. Here, the method SHOW-MENU, which displays and manages MENU-STOP, is a specialization of a more general method of the class NODE. Hence, the method SHOW-MENU in the class STOP handles specific selections about stops and refers to the method SHOW-MENU in the class NODE for more general selections, like MODIFY-COORDINATES.

This small example shows how the hierarchical organization of nodes allow a judicious partition of the attributes based on semantic considerations. As a matter of fact, stops and origins carry a special meaning in a transportation network, and the concept of specialization allows us to avoid redundancy by designing classes that encapsulate variables and methods specifically designed for these special kinds of nodes.

Example 2. The Classes VEHICLE and ROUTE

Creating routes for vehicles is the ultimate goal of ALTO and, consequently, the characteristics of those vehicles are very important. The value of some vehicle's IVs like CAPACITY, SPEED, MAXIMAL-DISTANCE and MAXIMAL-TRAVEL-TIME, constrains the set of feasible routes that can be attached to them (cf. description of the classes ROUTE and VEHICLE in figure 4).

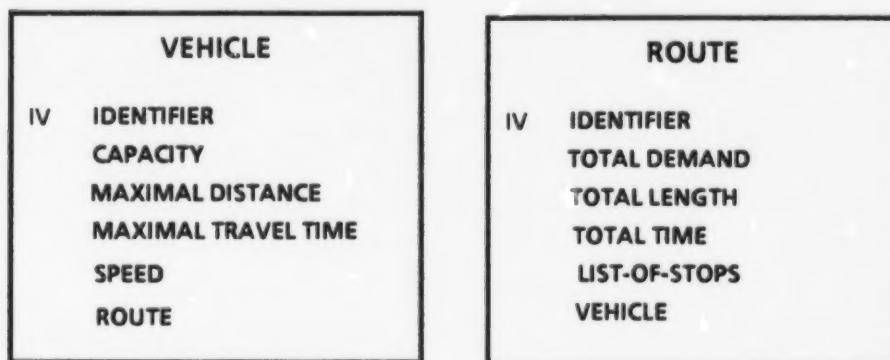


Fig. 4 Classes VEHICLE and ROUTE

Representing each vehicle as an individual object allows great flexibility in the specification of the fleet of vehicles. For example, it is straightforward to create homogeneous fleets of vehicles (where all vehicles have the same characteristics), by supplying default values to the class VEHICLE. Alternatively, heterogeneous fleets of vehicles can be created via standard facilities allowing modifications to the content of any particular object. In addition, the concept of specialization can be useful for defining restricted classes of vehicles, like cars and trucks. In our application domain, however, vehicle's specialization is not really required since we are dealing with attributes, like speed and capacity, that are shared by all kinds of vehicles.

Management of the current solution is facilitated by the bidirectional link between a vehicle and its route (cf. instance variable ROUTE in the class VEHICLE and instance variable VEHICLE in the class ROUTE). This link provides a natural and simple way to check if any modification to a given route meets the specification of the vehicle servicing this route. In order to do that, ALTO simply compares "sibling IVs" like TOTAL-DEMAND and CAPACITY, TOTAL-LENGTH and MAXIMAL-DISTANCE, TOTAL-TIME and MAXIMAL-TRAVEL-TIME. Moreover, any modification to the characteristics of a given vehicle can easily be propagated to the appropriate route. For example, any modification to the speed of a given vehicle will trigger an update of the total time of the linked route (and the service time of all serviced stops, as one can deduce from the instance variable LIST-OF-STOPS in the class ROUTE).

5.2 Concluding remarks

The small examples of section 5.1 illustrate ways to take advantage of the object-oriented paradigm in the vehicle routing domain. In more general terms, the benefits of the object-oriented approach are the followings:

1) Implementation issues ("internal" point of view)

- a) It is possible to introduce as many attributes as desired in a given class. For example, a vehicle is described by various attributes like speed, capacity, maximal travel time, etc. Obviously, this set can be augmented to suit new applications with some specific requirements.
- b) The concept of specialization in a multiple inheritance lattice allows the system to deal very easily with different types of nodes in a given network. The specialization of nodes is a fundamental concept in transportation problems for which object-oriented programming is very well suited.
- c) Default values can be used to characterize typical instances of a given class. They are used, for example, for creating homogeneous fleets of vehicles.
- d) Management of the current solution is facilitated by the object-oriented approach. As mentioned in section 4, three generic operators are used to create routes for vehicles. These operators are acting upon routing objects by using insertion and exchange heuristics. Implementation of the general heuristic is then greatly simplified by the fact that all variables and methods needed to update the current problem state are available through well-defined individual entities. For example, insertion of a new stop into an existing route by an instance of operator ADD will modify the set of stops, the length and the travel time of this route. Since the relevant informations are available in the same single instance of the class ROUTE, the updating process is therefore greatly facilitated.
- e) Mixins (and classes in general) are useful to encapsulate variables and methods providing some well-identified facilities (cf. mixins related to the class NETWORK). This is of great help when building large programs like ALTO.
- f) Extensions to other domains involving networks can easily be foreseen. For example, many parts of the program can be used, with just slight modifications, for "computer network" types of problems. To this end, the class NODE must be specialized according to the specific nature of the nodes in such networks (e.g. Hosts, IMPs, etc ...).

2) User interface issues ("external" point of view)

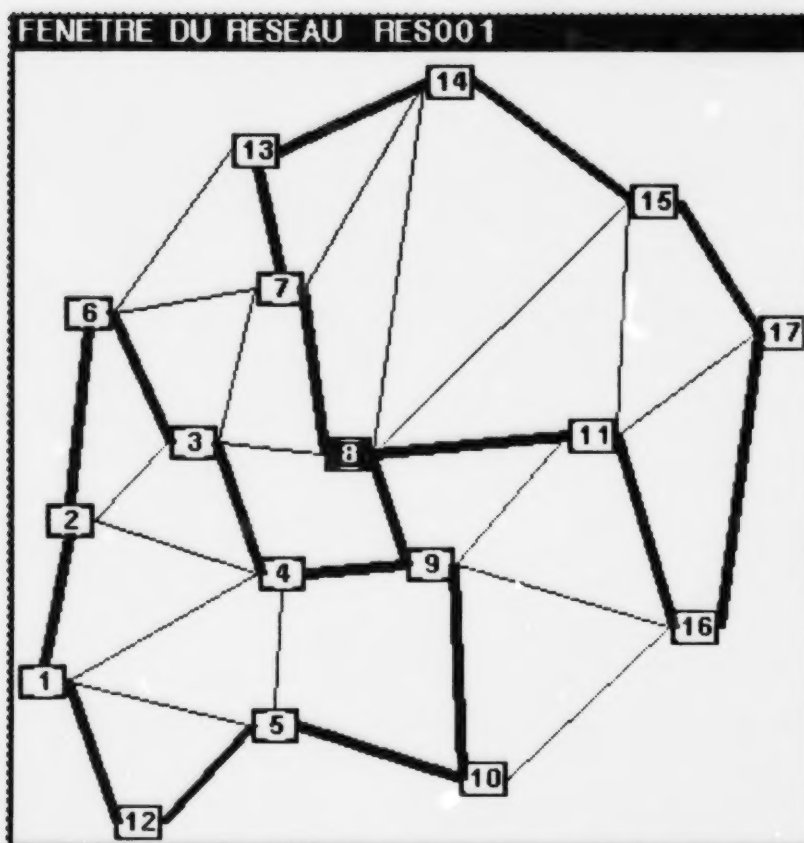
- a) Defining a vehicle routing problem via a set of well-defined classes of objects looks natural to people working in this area, since they are working with well-defined entities (stops, vehicles, etc...) to which they are accustomed to.
- b) The process of creating resolution strategies is also greatly facilitated, because the basic resolution mechanism of each generic operator is quite easy to understand. In fact, each operator applies primitive operations, like insertion, to sets of well-defined objects. This way of specifying vehicle routing algorithms looks natural to human beings and was very succesful in the pedagogical applications of the system (ALTO is used at Montreal University for teaching "classical" vehicle routing algorithms to graduate students)
- c) We must finally observe that INTERLISP-D allows creation of "inspection windows" displaying the content of any selected object (see figure 5). This feature adds a "distributed data base" aspect to the system. It is quite easy to implement such a feature because each object is an individual entity enclosing all relevant attributes. With this facility, it is possible to monitor the status of any object involved in the problem's definition (e.g. the current length of the routes, the service time at a given stop, etc...). Obviously, this information can guide the expert user when he/she is designing a new problem-solving strategy (i.e when he/she is creating formulas for instantiating the generic operators).

Hence, there is no doubt that the object oriented paradigm adds flexibility and user-friendliness to our system. Some of its capabilities could have been implemented with procedural languages, but it would have been much more difficult to do so because standard languages do not provide such a natural approach as the object-oriented paradigm for modeling and solving vehicle routing problems.

Section 6. Conclusion

In this paper, we described how object-oriented programming has been used in the development of a vehicle routing computer system, aimed at facilitating the design of new algorithms. The program is built around an inheritance lattice of classes of objects which are used to specify vehicle routing problems and to solve them. The object-oriented paradigm is very attractive in such a context, because vehicle routing problems can be easily decomposed into well-defined entities, and specification of user-defined resolution strategies is seen accordingly as a process of instantiating basic problem-solving classes (generic operators). Furthermore, such a representation facilitates future extensions to the current system, either by extending its target class of routing problems or by adapting it to other kinds of network problems. Current

developments are aimed at implementing a "commercial" version of the system on a Macintosh microcomputer. We are also considering the inclusion of mixed pick-up and delivery problems (dial-a-ride types of problems, see [1]).



NOEUD 7

NUMERO	7
IDENTIFICATEUR	N0017
COORDONNEES	(289 446)
VEHICULE	V2
DEMANDE	100
TDEM	2
TOPT	25
FENETRE-TEMPS	(10 50)
TEMPS	2.2277

ARC (7 13)

NUMERO	45
IDENTIFICATEUR	A0065
NOEUDSOURCE	N0017
NOEUDCIBLE	N0021
LONGUEUR	100.62
ANGLE	101
RUE	NIL
INTERDICTION	NIL

Fig 5. Inspection windows for a node and a link

References

- [1] Bodin L., Golden B., Assad A. and Ball M., "Routing and Scheduling of Vehicles and Crews: The State of the Art", *Computers & Operations Research*, vol. 10, no. 2 (1983).
- [2] Lapalme G., Potvin J.Y. and Rousseau J.M. , "A General Heuristic for Node Routing Problems", in *Advances in Optimization and Control*, H.A. Eiselt & G. Pederzoli eds, Springer-Verlag, pp. 124-143 (1988).
- [3] Potvin J.Y., Lapalme G. and Rousseau J.M., "ALTO: A Computer System for the Design of Vehicle Routing Algorithms", *Computers & Operations Research*, vol.16, no. 5, pp. 451-470 (1989).
- [4] Stefik M., Bobrow D., Mittal S. and Conway L., "Knowledge Programming in LOOPS", *The AI Magazine*, pp. 3-13 (1983).
- [5] Stefik M. and Bobrow D., "Object-Oriented Programming: Themes and Variations", *The AI Magazine*, pp. 40-62 (1986).
- [6] Teitelman W. and Masinter L., "The Interlisp Programming Environment", *Computer*, pp. 25-32 (1981).